Algorithms: Tours in Graphs

Amotz Bar-Noy

CUNY
Outline

1. Introduction
2. Eulerian Paths and Eulerian Cycles
3. De-Bruijn Sequences and Graphs
4. Hamiltonian Paths and Hamiltonian Cycles
Special Paths and Cycles in Graphs

**Eulerian Path**
- A path that traverses **all** the edges of the graph exactly once.

**Eulerian Cycle**
- A cycle that traverses **all** the edges of the graph exactly once.

**Hamiltonian Path**
- A simple path that visits **all** the vertices of the graph exactly once.

**Hamiltonian Cycle**
- A simple cycle that visits **all** the vertices of the graph exactly once.
An Eulerian Path
An Eulerian Cycle
A Hamiltonian Path
Complexity

Eulerian paths and cycles
- An Eulerian Path or a Cycle, if exist, can be found in any graph with $m$ edges in $\Theta(m)$-time.

Hamiltonian paths and cycles
- Finding a Hamiltonian Path or a Cycle is a hard to solve problem.
- It is strongly believed that no polynomial time algorithm exists to solve these problems.

The Traveller Salesperson Problem (TSP)
- The TSP is a generalization of the Hamiltonian Cycle problem to weighted graphs.
- There are “good” heuristics and approximation algorithms for weight functions that obey the triangle inequality.
- In particular for graphs that represent distances in the plane.
Eulerian Paths and Cycles in Undirected Graphs

Edge representation path

- \( P = (e_0, e_1, \ldots, e_{m-1}) \)

Definitions

- \( P \) is an Eulerian Path in an undirected graph with \( m \) edges if
  - \( e_i \neq e_j \) for all \( 0 \leq i \neq j < m \).
  - \( e_i = (x, y) \) and \( e_{i+1} = (y, z) \) for \( 0 \leq i < m - 1 \) and vertices \( x, y, z \).

- An Eulerian Cycle \( C \) in an undirected graph is an Eulerian Path \( P \) for which
  - \( e_{m-1} = (x, y) \) and \( e_0 = (y, z) \) for vertices \( x, y, z \).
Eulerian Paths and Cycles in Directed Graphs

Edge representation path

\[ P = (e_0, e_1, \ldots, e_{m-1}) \]

Definitions

- \( P \) is an Eulerian Path in a directed graph with \( m \) edges if
  - \( e_i \neq e_j \) for all \( 0 \leq i \neq j < m \).
  - \( e_i = (x \rightarrow y) \) and \( e_{i+1} = (y \rightarrow z) \) for \( 0 \leq i < m - 1 \) and vertices \( x, y, z \).
- An Eulerian Cycle \( C \) in a directed graph is an Eulerian Path \( P \) for which
  - \( e_{m-1} = (x \rightarrow y) \) and \( e_0 = (y \rightarrow z) \) for vertices \( x, y, z \).
The bridges of Königsberg

No Eulerian Cycle or Eulerian Path exist!!!
A Toy Example

- The **left** graph has no Eulerian Path.
- The **middle** graph has an Eulerian Path but not an Eulerian Cycle.
- The **right** graph has an Eulerian Cycle.
Graphs with Eulerian Cycles

Theorem

An undirected and connected graph has an Eulerian Cycle iff all the vertices have an even degree.

Remark

A self-loop adds 2 to the degree of the vertex.

Proof: the only-if direction

- Let $C = (e_0, e_1, \ldots, e_{m-1})$ be an Eulerian Cycle.
- Let $y$ be a vertex.
- If $e_i = (x, y)$ then $e_{i+1} = (y, z) \ ( (m - 1) + 1 = 0 )$.
- Therefore, the degree of $y$ must be even.
Outline

- Assume all the degrees are even.
- Construct an $\Theta(m)$-time algorithm producing an Eulerian Cycle represented by vertices.
- Each edge is examined constant number of times with an appropriate data structure.
- **Main idea:** Explore unused edges as long as they exist.
Construction

Data structure and variables

- Edges are marked either **used** or **unused**.
  - Initially all the edges are marked **unused**.
  - At the end all the edges are marked **used**.

- An arbitrary starting vertex \( x \).

- A main cycle \( C \).
  - Initially \( C \) is empty.
  - At the end \( C \) contains all the edges.

- An exploring path \( P = (y, \ldots) \).
  - Initially \( P = (x) \).
  - At the end \( P \) is empty.

- A secondary cycle \( C' \).
  - Initially and at the end \( C' \) is empty.
Finding a secondary cycle

- Let \( P = (y, \ldots, z) \) be the exploring path.
- While \( z \) (the last vertex in \( P \)) has unused edges:
  - Let \((z, w)\) be an unused edge.
  - Mark \((z, w)\) as used.
  - Append \( w \) at the end of \( P \): \( P = (y, \ldots, z, w) \).
- Let the secondary cycle \( C' = P = (y, \ldots, y) \).
  - Need to prove: this process terminates only at \( y \).
Combining the main and the secondary cycles

- Let $C = (x, \ldots, a, y, b, \ldots, x)$ be the main cycle.
- Let $C' = (y, c, \ldots, d, y)$ be the secondary cycle.
- Then $C = (x, \ldots, a, y, c, \ldots, d, y, b, \ldots, x)$. 
Construction

High level algorithm

1. Start the exploring path with \( x \).
2. Find the first secondary cycle \( C' \):
3. Set the first main cycle \( C \) to be \( C' \).
4. While there exists an unused edge:
   4.1. Find \( y \) in \( C \) with an unused edges.
   4.2. Find a secondary cycle \( C' \) starting with \( y \).
   4.3. Combine the cycles \( C \) and \( C' \) into \( C \).
5. Return the cycle \( C \).
Correctness

Observation I

Since all the edges have an even degree it follows that the finding a secondary cycle procedure can be stuck only at \( y \) which is the first vertex of the exploring path.

Observation II

If the main cycle \( C \) does not contain all the edges in the graph, then it must contain a vertex with an unused edge due to connectivity.

Correctness proof

Followed by the above two observations.
Complexity

Outline

- Each edge is explored only once when it is \textit{unused} and then it becomes \textit{used} forever.
- This can be done in $\Theta(m)$-time with \textit{adjacency lists}.
- Each edge is traversed only once while looking for a vertex with an \textit{unused} edge in the main cycle.
- This can be done if the main cycle is a \textit{linked list} and if the algorithm \textit{maintains} the last starting vertex for the exploring path.
- There are at most $n - 1$ cycle combinations since a new exploring path never reaches again the connecting vertex.
- A combination can be done in $\Theta(1)$-time if the cycles are maintained as \textit{double linked lists}.
- The overall complexity is $\Theta(m)$ in connected graphs ($n \leq m$).
Directed Graphs

Definitions

- In a **strongly connected graph** there exists a directed path between any two vertices.
- The **in-degree** of a vertex \( x \) is the number of edges terminating at \( x \).
- The **out-degree** of a vertex \( x \) is the number of edges originating at \( x \).

Theorem

- A directed and strongly connected graph has an Eulerian Cycle **iff** \( d_{in}(x) = d_{out}(x) \) for each vertex \( x \).
Eulerian Paths

**Theorem: undirected graphs**

- An undirected and connected graph has an Eulerian Path if at most two vertices have an odd degree.

**Theorem: directed graphs**

- A strongly connected directed graph has a directed Eulerian Path starting with $x$ and ending at $y$, $x \neq y$, if:
  - $d_{in}(z) = d_{out}(z)$ for any vertex $z \notin \{x, y\}$.
  - $d_{in}(x) = d_{out}(x) - 1$.
  - $d_{in}(y) = d_{out}(y) + 1$. 

Amotz Bar-Noy (CUNY)
Covering Paths

**Lemma**
- In an undirected graph the number of vertices with odd degree is even.

**Definition**
- $k$ disjoint paths **cover** a graph $G$ if each edge of $G$ belongs to one of the $k$ paths.

**Theorem**
- A connected undirected graph with $2k$ vertices with an odd degree can be **covered** with $k$ disjoint paths.
Covering Paths

Proof outline

- Match the odd-degree $2k$ vertices with $k$ new edges.
  - The original and the new graphs do not have to be simple.
- All the vertices in the new graph have now an even degree.
- Find an Eulerian Cycle in the new graph.
- The new edges are not adjacent in the Eulerian Cycle since each vertex belongs to at most one new edge.
- Omit the $k$ new edges from the Eulerian Cycle.
- The cycle is partitioned to $k$ paths that cover all the original edges.
De-Bruijn Sequences

Notation
- $\Sigma = \{0, 1, \ldots, \sigma - 1\}$ – an alpha-bet of $\sigma$ letters.

Observation
- There exists $\sigma^\ell$ distinct words of length $\ell$ over $\Sigma$.

Examples
- $\sigma = 2$ and $\ell = 3 \implies \{000, 001, 010, 011, 100, 101, 110, 111\}$
- $\sigma = 3$ and $\ell = 2 \implies \{00, 01, 02, 10, 11, 12, 20, 21, 22\}$
**Definition**

A cyclic sequence

\[ S_{\sigma, \ell} = a_0, a_1, \ldots, a_{L-1} \]

of length \( L = \sigma^\ell \) is called a **De-Bruijn** sequence if for any word \( w \) of length \( \ell \) over \( \Sigma \) there exists a unique index \( 0 \leq i < L \) such that

\[ w = a_i, a_{i+1}, \ldots, a_{i+\ell-1} \]

where the addition is done mod \( L \).

**Examples**

- \( \sigma = 2 \) and \( \ell = 3 \) \( \implies \) (00011101)
- \( \sigma = 3 \) and \( \ell = 2 \) \( \implies \) (001122021)
Directed De-Bruijn graphs

Graph

- Denote a De-Bruijn graph by $G_{\sigma,\ell} = (V_{\sigma,\ell}, E_{\sigma,\ell})$.

Vertices

- All the $n = \sigma^{\ell-1}$ words of length $\ell - 1$.
  - $V_{2,4} = \{000, 001, \ldots, 111\}$.
  - $V_{3,3} = \{00, 01, \ldots, 22\}$.

Edges

- All the $m = \sigma^\ell$ words of length $\ell$.
  - $E_{2,4} = \{0000, 0001, \ldots, 1111\}$.
  - $E_{3,3} = \{000, 001, \ldots, 222\}$.

- The edge $(b_1, \ldots, b_{\ell})$ connects the vertices:
  $(b_1, b_2 \ldots, b_{\ell-1}) \rightarrow (b_2, \ldots, b_{\ell-1}, b_{\ell})$
De-Bruijn Sequences and Graphs

$G_{2,3}$

Graph $G_{2,3}$ showing tours in graphs.
$G_{3,2}$
De-Bruign Sequences Always Exist

**Lemma**
- For all positive integers $\sigma$ and $\ell$ there exists a directed Eulerian Cycle in $G_{\sigma,\ell}$.

**Proof**
- $G_{\sigma,\ell}$ is strongly connected and $\text{in-degree} = \text{out-degree} = \sigma$ for all vertices.

**Lemma**
- An Eulerian Cycle in $G_{\sigma,\ell}$ implies a De-Brujin sequence $S_{\sigma,\ell}$.

**Proof**
- Follow the Eulerian Cycle. Initially the sequence is the first vertex on the path. Append only the last letter of the next vertex to the current sequence.
**$G_{2,3}$ and $S_{2,3}$**

- **Eulerian Cycle:** $00 \rightarrow 00 \rightarrow 01 \rightarrow 11 \rightarrow 11 \rightarrow 10 \rightarrow 01 \rightarrow 10 \rightarrow 00$
- **De-Bruijn sequence:** $00011101$
$G_{3,2}$ and $S_{3,2}$

**Eulerian Cycle:** $0 \rightarrow 0 \rightarrow 1 \rightarrow 1 \rightarrow 2 \rightarrow 2 \rightarrow 0 \rightarrow 2 \rightarrow 1 \rightarrow 0$

**De-Bruijn sequence:** 001122021
Hamiltonian Paths and Cycles

Undirected graphs

- A path of vertices: \( P = (v_0, v_1, \ldots, v_{n-1}) \).
- \( P \) is a Hamiltonian Path in a graph with \( n \) vertices if
  - \( v_i \neq v_j \) for all \( 0 \leq i \neq j < n \).
  - \((v_i, v_{i+1})\) is an edge for \( 0 \leq i < n - 1 \).
- A Hamiltonian Cycle \( C \) is a Hamiltonian Path \( P \) for which
  \((v_{n-1}, v_0)\) is also an edge.

Directed graphs

- A directed path of vertices: \( P = (v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_{n-1}) \).
- \( P \) is a directed Hamiltonian Path in a graph with \( n \) vertices if
  - \( v_i \neq v_j \) for all \( 0 \leq i \neq j < n \).
  - \((v_i \rightarrow v_{i+1})\) is a directed edge for \( 0 \leq i < n - 1 \).
- A directed Hamiltonian Cycle \( C \) is a directed Hamiltonian Path \( P \)
  for which \((v_{n-1} \rightarrow v_0)\) is also an edge.
Hamiltonian Paths and Hamiltonian Cycles

The Petersen Graph

Observations

- There is no Hamiltonian Cycle.
- The following is a Hamiltonian Path:
  \[ P = (A, B, C, D, E, J, H, F, I, G) \]
The Knight-Chess Graph

Definition

- The **Knight-Chess graph** has \( n^2 \) vertices; one for each square on the \( n \times n \) chess board. Two vertices are adjacent iff a knight can move from one to another in one step.

The \( 8 \times 8 \) Knight-Chess graph
The Knight-Chess Problem

Problem

- Is it possible to cover all the squares of the chess board with knight moves?
- An equivalent formulation: Are there Hamiltonian paths in Knight-Chess graphs?

The $8 \times 8$ Knight-Chess graph
### Definition

- A **tournament** is a simple directed graph such that for each pair of vertices $u$ and $v$, either the directed edge $u \rightarrow v$ exists or the directed edge $v \rightarrow u$ exists but not both and not none.

### Observations

- There are exactly $\binom{n}{2}$ directed edges in a tournament with $n$ vertices.
- The underlying graph of a tournament with $n$ vertices is the complete graph $K_n$.

### Theorem

- A tournament always has a Hamiltonian path.
A Tournament with 6 Vertices
A Hamiltonian Path in the Tournament
A Hamiltonian Cycle in the Tournament
Finding a Hamiltonian Path in a Tournament

High level algorithm

1. **Start** with the path \( P_1 = (v_1) \) for an arbitrary vertex \( v_1 \).
2. For \( 1 \leq i \leq n \), let the current path be
   \[
   P_i = (v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_i)
   \]
3. If \( i = n \), **terminate** with the Hamiltonian Path \( P_n \).
4. Let \( v \) be a vertex not in the path.
5. **Insert** \( v \) into \( P_i \) to get the path \( P_{i+1} \).
6. **Goto** Step (2).
Path Augmentation

Three cases

- If \((v \rightarrow v_1)\) is an edge, then \(P_{i+1} = (v \rightarrow v_1 \rightarrow \cdots \rightarrow v_i)\).
- If \((v_i \rightarrow v)\) is an edge, then \(P_{i+1} = (v_1 \rightarrow \cdots \rightarrow v_i \rightarrow v)\).
- Otherwise, \(\exists 1 \leq j < i\) s.t. \((v_j \rightarrow v)\) and \((v \rightarrow v_{j+1})\) are edges, then
  \[
P_{i+1} = (v_1 \rightarrow v_j \rightarrow v \rightarrow v_{j+1} \cdots \rightarrow v_i)
  \]
Correctness and Complexity

Correctness

- The path augmentation is always successful.
- Therefore, eventually $P_n$ exists which is a Hamiltonian Path.

Complexity

- Inserting a vertex to a path can be done in $\Theta(n)$ time using the adjacency matrix.
- There are $n$ iterations.
- The overall complexity is $\Theta(n^2)$.
- With a binary search for the insertion point, the algorithm probes the adjacency matrix $O(n \log n)$ times. But the overall complexity is still $\Theta(n^2)$. 
A Hamiltonian Cycle Greedy Algorithm

High level algorithm

- As long as possible, **construct** a path by adding vertices to both end-vertices of the path.
- **Close** this path into a cycle by either connecting both end-vertices or by finding a **switch vertex**.
- **Connect** a new vertex to the cycle and **break** it to be a new longer path.
- **Repeat** the above process until either a Hamiltonian Cycle is found or an operation is impossible.
Converting a Path to a Cycle
Converting a Cycle to a Path
A Hamiltonian Cycle Greedy Algorithm

Algorithm part I

1. Initially, let $P = (x)$ be a path with an arbitrary vertex $x$.
2. **Expand** the path $P$ from both ends until impossible. Let
   
   $P = (x_0 - x_1 - \cdots - x_h)$

   where there are no edges from $x_0$ and $x_h$ outside $P$.
3. If $(x_0, x_h)$ is an edge then **construct** the cycle
   
   $C = (x_0 - x_1 - \cdots - x_h - x_0)$

   **Goto** step 6.
4. If for some $0 < i < h$ the edges $(x_0, x_{i+1})$ and $(x_i, x_h)$ exist, then **construct** the cycle
   
   $C = (x_0 - x_1 - \cdots - x_i - x_h - x_{h-1} - x_{i+1} - x_0)$

   **Goto** step 6.
A Hamiltonian Cycle Greedy Algorithm

Algorithm part II

5. **Terminate Unsuccessfully** with the path $P$.

6. If $h = n - 1$ then **Terminate Successfully** with the Hamiltonian Cycle $C$.

7. If there is no edge from $C$ outside of $C$, then **Terminate Unsuccessfully** with the cycle $C$.

8. Let $(x_i, x)$ be an arbitrary edge from $C$ to outside of $C$, then **construct** the path

$$P = (x - x_i - x_{i+1} - \cdots - x_h - x_0 - \cdots - x_{i-1})$$

9. **Goto** step 2 with a longer path.
Sometimes Hamiltonian Cycles Exist

**Theorem**
- Let $G$ be a connected graph with $n$ vertices.
- If $d(u) + d(v) \geq n$ for any two vertices $u \neq v$ in $G$, then $G$ has a Hamiltonian Cycle.

**Corollary**
- Let $G$ be a connected graph with $n$ vertices.
- If $d(u) \geq n/2$ for any vertex $u$ in $G$, then $G$ has a Hamiltonian Cycle.
Proof of the Theorem

Proof outline

- Step 4, whenever executed, is always successful.
  - Assume that step 4 fails for $h \leq n - 1$ with the path $P = (x_0 - x_1 - \cdots x_{h-1} - x_h)$.
  - Let $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ be the neighbors of $x_0$ in $P$.
  - $\Rightarrow x_{i_1-1}, x_{i_2-1}, \ldots, x_{i_k-1}$ cannot be neighbors of $x_h$.
  - $\Rightarrow d(x_h) \leq h - k \leq n - 1 - k$.
  - $\Rightarrow d(x_0) + d(x_h) < n$.
  - A contradiction.

- Therefore, the algorithm never reaches step 5.
- The algorithm never terminates in step 7 since the graph is connected.
- The algorithm terminates successfully with a Hamiltonian Cycle in step 6 since the path is longer in each iteration.
Algorithm Complexity

Outline

- Represent the graph with an adjacency matrix.
- Augmenting a path by one vertex at its end-point can be done in $\Theta(n)$-time for a total of $\Theta(n^2)$-time for all the augmentations.
- Converting a path into a cycle can be done in $\Theta(n)$-time for a total of $\Theta(n^2)$-time for all such conversions.
- All the conversions of cycles into paths can be done in $\Theta(n^2)$-time by scanning the adjacency matrix only once.
- The overall complexity is therefore $\Theta(n^2)$. 